Chapter

1

Introduction to Languages and Notations

This chapter describes the benefits of formal methods. It also gives a brief introduction to the TTCN, the ASN.1 and the MSC language.

Note that this chapter is not a tutorial on TTCN, ASN.1 or MSC. In TTCN Suite Methodology Guidelines you can find more information about TTCN and how to use it.

If you want to know more about the languages supported in the SDL suite, you should read chapter 1, *Introduction to Languages and Notations, in the SDL Suite Getting Started*.

Standardized Formal Methods

It is getting increasingly accepted within a steadily growing range of industrial segments that the only true way for software engineering to achieve higher quality, deliver on time and decrease development costs, is to use formal methods. Furthermore, as the international market grows, equipment from different manufacturers must be able to communicate with each other. Therefore it is obvious that the formal method to be used should be internationally standardized.

There are a number of different formal standardized languages and methods available today. The one you select, however, should fulfill a few more important requirements.

One is the availability of professional development tools. Another is clarity. The notation should be understood by a general audience, from experts to end users. Ideally it should have a graphical syntax.

Formal standardized graphical languages ...

- ... enforce precision during specification, since ambiguities and unclear statements are impossible to make.
- ... allow tool support. Tools can help to perform all kind of analyses like syntax/semantics check, simulation, test generation, code generation, etc.
- ... attract more than one tool-builder. The second-source possibility creates vendor independence with all its advantages.
- ... are more often the subject for courses, seminars, and text-books.
- ... are more likely to be maintained. As with natural languages, formal languages need to evolve to stay modern.
- ... promotes efficient verbal and written communication within development teams, between manufactures and between suppliers and customers, due to the conceptually formalized means of communication.
- ... have a graphical syntax that makes it simple and efficient to exchange information between different players within an organization.

The Test Suite Framework Standard

As the use of standards within the world of Information Technology and Telecommunications has increased tremendously during the last decade, so has the need for methods and tools that support the verification and validation of both the standards and their implementations.

This need has been addressed by ISO and CCITT (ITU-T) in the "Framework and Methodology for Conformance Testing of Implementations of OSI and CCITT Protocols". The framework has now reached the status of an International Standard as ISO/IEC 9646 (or X.290).

- The standard introduces the concept of *Abstract Test Suites* (consisting of *Abstract Test Cases*), a description of a set of tests that should be executed for a system. The tests should be described using a black-box model, i.e. only control and observe using the available external interfaces.
- The abstract tests are to be described using a formal language rather than using informal natural language. As part of the standard, the language TTCN is defined in order to describe the abstract tests.
- The possibility to copy the ASN.1 definitions from the protocol specification into the test suite in TTCN assures consistency between the information transferred in system specification and the test specification.

Conformance Testing

Conformance testing is the process of verifying that an implementation performs in accordance with a particular standard/specification/environment.

Conformance testing is exclusively concerned with the external behavior of an implementation. Service and functional behavior is tested in order to find logical errors and prerequisites for interoperability. Conformance testing is not intended to be exhaustive and a successfully passed test suite does not imply a 100% guarantee. But it does ensure, with a reasonable degree of confidence, that the implementation is consistent with its specifications, and it does increase the probability that implementations will interwork.

System Testing

- *Conformance testing* verifies whether an implementation performs according to the stated standard/specification/environment.
- *Interoperability testing* checks the ability of different implementations to interact in a prescribed manner, achieving predictable results.
- *Regression testing* is performed after functional improvements or corrections, to confirm that nothing unintentional has been introduced.

The TTCN Language

TTCN (Tree and Tabular Combined Notation, ISO/IEC 9646-3) is a language standardized by ISO for the specification of tests for real-time and communicating systems. TTCN has been developed within the framework of standardized conformance testing (ISO/ IEC 9646).

With TTCN a test suite is specified. A test suite is a collection of various test cases together with all the declarations and components it needs. Each test case is described as an event tree. In this tree, behaviors such as “First, we send A, then either B or C is received; if it was B we will send D...” are described. Concurrent TTCN allows several event trees to run concurrently.

TTCN is abstract in the sense of being test system independent. This means that a test suite in TTCN for one application (e.g. protocol, system, etc.) can be used in any test environment for that application.

The use of TTCN has increased tremendously during the last few years. This has been augmented by the significant amount of test suites released by various standardization bodies. TTCN is not only used in standardization work. The language is very suitable for all kinds of functional testing for real-time and communicating systems. This has led to a wide usage throughout the industry.

The specifications of the messages being sent and received can be defined using either the native form of TTCN or by using ASN.1 (Abstract Syntax Notation One).

Theoretical Model

A TTCN specification describes an abstract test suite (ATS) that is independent of test system, hardware and software. The ATS defines the test of the *implementation under test* (IUT), which is treated in a black box model, i.e. only its exterior interface is of concern. The IUT is stimulated by sequences of test events and its response is inspected.

A TTCN abstract test suite can be transformed into an *executable test suite* (ETS) using the TTCN suite. This ETS is downloaded into the test system (the system performing the test).

The test system performs the test by executing the ETS against the *system under test* (SUT) which contains the implementation under test.

During execution the ETS will report any errors and log events for on-line or post-test evaluation.

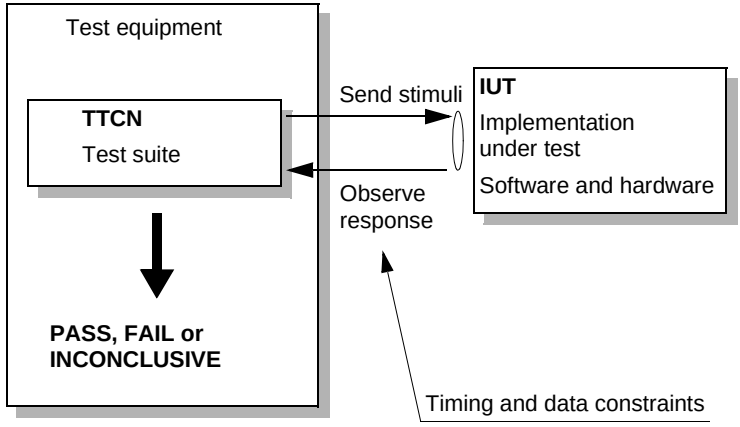

Figure 1: Test System with Executable Test Suite (ETS) connected to the system under test (SUT)

TTCN Specification Structure

A TTCN specification is similar to a Pascal or C program. (Being cleaner and more comprehensive, TTCN is easier to learn.) Just like Pascal and C, TTCN requires type and data declarations and it uses concepts like modules and subroutines. Since TTCN is designed for testing, it contains test specific concepts such as:

- Powerful pattern matching constructs for complex data structures using both TTCN and ASN.1.
- Verdicts and preliminary verdicts to define the outcome of test cases.
- Possibilities to handle alternative outcomes in a test case.
- Pre-ambles and post-ambles to show how to compose test cases.
- The “modules” concept supporting multi-user test development.
- The “modules” concept supporting the re-use of test components and data structures.
- Constructs for parallel test component execution including synchronization primitives.

A TTCN specification has a standardized layout that produces comprehensive and unambiguous paper printouts. This greatly improves clarity and readability. A test suite is divided into the following four major parts:

- The *overview part*, containing a table of contents and a description of the test suite. Its purpose is mainly to document the test suite to increase clarity and readability.
- The *declarations part*, declaring all messages, variables, timers, data structures and black box interface towards the Implementation Under Test.
- The *constraints part*, assigning values and creating constraints for inspection of responses from the implementation under test.
- The *dynamic part*, containing all test cases, test steps and default tables with test events and verdicts, i.e. it describes the actual execution behavior of the test suite.

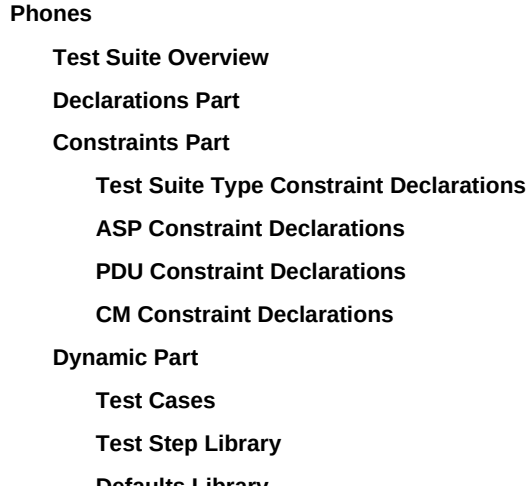

Figure 2: The basic structure of a TTCN Test Suite

Test Suite Dynamic Structure

The dynamic part of a TTCN abstract test suite is created in a hierarchical and nested manner. The building blocks are test groups, test cases, test steps and test events. There are no limitations as to how many test groups may be contained in a test suite, how many test events may be contained in a test step, etc.

Test component explanation:

- *Test event:* The smallest, indivisible unit of a test suite. Typically, it corresponds to a signal, interrupt, message, data or timer expiration.
- *Test step:* A grouping of test events, similar to a subroutine or procedure in other programming languages.
- *Test case:* The main fundamental building block in a test suite. A test case tests a particular feature or function in the implementation under test (IUT). A test case has an identified test purpose and it assigns a verdict that depends on the outcome of the test case.
- *Test group:* A grouping of test cases. It might for example be convenient to group all test cases concerning connection establishment, and to put all test cases concerning transport into a separate test group.
- *Test suite dynamic part*: The highest level, encompassing all test components and serving as the root of the tree. A test suite can range from a large number of test groups and test cases to a single test event contained in a test case.

Dynamic Part

Test Cases

BasicCall

CallW

Test Step Library

ConnectPhones

TestCallWaiting

HangUpAllPhones

Defaults Library

Figure 3: The TTCN dynamic part structure

Communication Mechanisms

TTCN uses the concepts of *points of control and observation* (PCOs), *abstract service primitives* (ASPs) and *protocol data units* (PDUs) in order to create an abstract interface towards the implementation under test (IUT). A PCO is a point in the abstract interface where the IUT can be stimulated and its responses can be inspected. An ASP or a PDU is either a stimuli or a response that carries information, i.e. parameters and data.

Each PCO has two *first in first out* queues for temporary storage of ASPs and PDUs: One queue for send and one queue for receive. These queues are infinite, i.e. they can store any number of ASPs and PDUs.

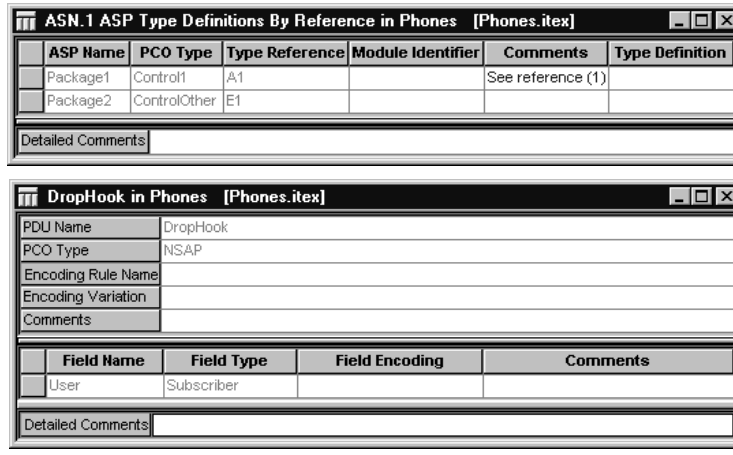

Figure 4: PCOs together with ASPs and PDUs create an abstract interface towards the IUT

Event Trees, Constraints and Verdicts

TTCN uses event trees with test events to express the behavior of test steps and test cases.

All the leaves in the event tree are assigned a verdict that can be PASS, FAIL or INCONCLUSIVE. PASS means that the test case completed without detecting any error. FAIL means that an error was detected, that is, the behavior of the IUT did not conform with the pre-defined specification. INCONCLUSIVE means that there was insufficient evidence

for a conclusive verdict to be assigned, but that the behavior of the IUT was valid.

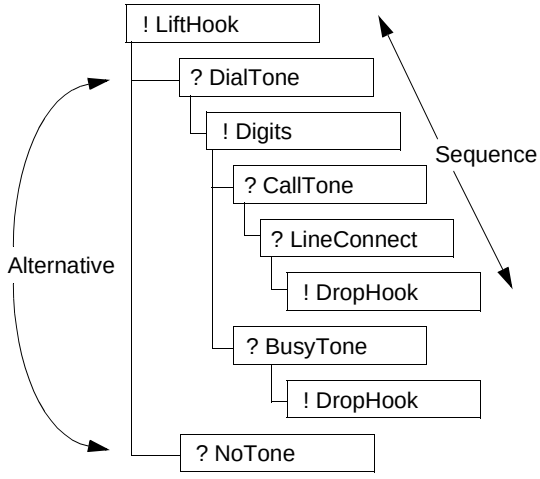

Figure 5: An event tree and the corresponding execution order

A verdict can be either preliminary or final, allowing for flexibility in the specification. A final verdict will terminate the active test case and return its verdict. A preliminary verdict will not terminate test case execution but it will flag either PASS, FAIL or INCONCLUSIVE. This preliminary verdict can be inspected during test execution, like any variable.

TTCN uses the comprehensive format shown in Figure 6. The indentation level of statements in the Behaviour Description column indicates where in the event tree an event belongs. The leaves of the tree hold verdicts that define test case outcome.

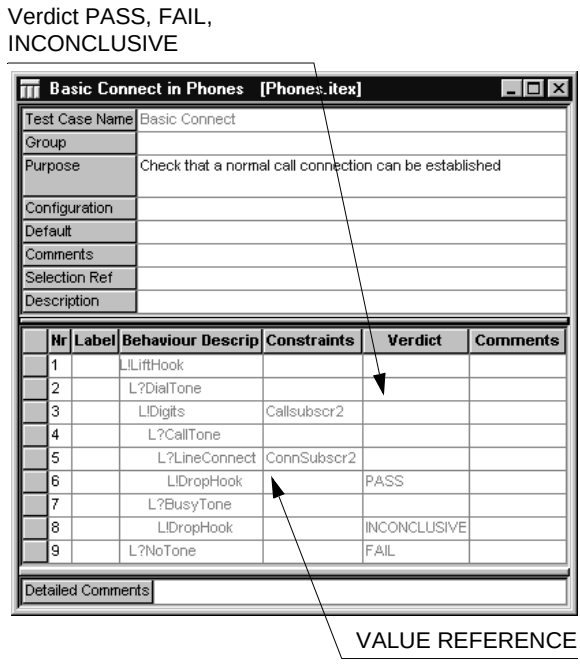

Figure 6: The TTCN test case corresponding to the event tree in Figure 5

Data and Value Model

The TTCN data and value model is somewhat different from the one in traditional programming languages. It allows for the creation of complex data structures and types, and has the concept of constraints to do value assignments. Constraints are more powerful than values in that they also allow the use of patterns. A pattern can contain wild cards and define allowable value ranges for complex data structures. This is very useful when inspecting responses from the implementation under test.

TTCN has two alternative data and constraint representation formats: the TTCN native tabular form and ASN.1 (Abstract Syntax Notation One). ASN.1, which is a purely textual notation, provides a more flexible platform for describing complex data structures. ASN.1 also allows data descriptions to be shared between an SDL (Z.105) specification and a TTCN test suite.

Modular TTCN

With modular TTCN, it is possible to define test suite components for re-use. This facilitates test component re-use, and provides a language platform for multi-user test development projects. See also "Distributed Development (UNIX)" on page 118 in chapter 1, *The TTCN Introduction, in the TTCN Suite Methodology Guidelines*.

Concurrent TTCN

Concurrent TTCN introduces a parallel architecture for simultaneous execution of several test components, allowing many interfaces to be tested concurrently. There are several benefits:

- Test components become more cohesive/modular since they focus on the test of a specific interface of the IUT. Each interface is isolated in specific test components.
- Module and integration testing is easier and it facilitates test component re-use.
- Test suite maintenance becomes easier since test components are less monolithic. If one interface of the IUT changes, it will not influence any test components but the ones specifically dedicated to this interface.
- Each test component becomes smaller and simpler since it deals with fewer alternatives.

In concurrent TTCN, each test case consists of several *parallel test components* (PTCs) that execute autonomously, performing concurrent tests. A *master test component* (MTC) starts the execution of the PTCs and controls the final verdict. The PTCs can only set preliminary verdicts and the test case is completed and its verdict is decided when all the PTCs are finished. The PTCs are synchronized through *co-ordination points* (CPs) and *co-ordination messages* (CMs).

Graphical and Textual Notations

The TTCN language supports two notations that are equivalent. The graphical notation (TTCN-GR) and a textual notation (TTCN-MP).

Application Areas

Currently, TTCN is mainly known within the telecommunications industry. However, it has broader areas of application, which can be summarized as follows:

- Any protocol conformance testing.
- Any communicating systems testing, e.g. interactive, message-driven, real-time or distributed.
- Any system with a well defined interface that can be stimulated and observed.

The ASN.1 Language

Abstract Syntax Notation One is a language specifically designed for describing structured information that is conveyed across some interface or communication medium. ASN.1 is standardized internationally (ISO/IEC 8824) and it is a key ingredient of Open Systems Interconnection (OSI).

In the presentation layer of the OSI hierarchy, data values of quite complex types, such as character strings, intricate structures or arrays of values, need to be determined in a unique way without saying anything about the representation. ASN.1 is developed to fill this need.

ASN.1 is a generic notation for the specification of data types and values. The basic principle is to define a small number of simple types by defining their possible values, and give rules for combining these into increasingly complicated types. The original use of ASN.1 was in the information description of high-level protocols (FTAM, CMIP, MHS etc.), but today it is widely used in the telecommunications industry for protocols and applications.

```
AtmInterfaceTCEntry     ::= SEQUENCE {
atmInterfaceOCDEvents       Counter32,
atmInterfaceTCAlarmState    INTEGER
}
```

Figure 7: A sample ASN.1 type definition

ASN.1 Encoding and Transfer Syntax

ASN.1 requires a transfer syntax in order to pass data between two entities. *Basic encoding rules* (BER, ISO 8825) is a standardized transfer syntax of OSI. Others exist as well: *canonical encoding rules* (CER) for security applications, *distinguished encoding rules* (DER) for digital signatures, traditional C/C++, etc. Any transfer syntax can be used for ASN.1 descriptions.

ASN.1, SDL and TTCN – a Powerful Combination

TTCN includes ASN.1, i.e. ASN.1 is used for creating data descriptions and constraints in test suite specifications. Through the new standard of Z.105, ASN.1 is merged with SDL (Specification and Description Language) to create an extremely powerful language environment for specification of real-time, interactive and distributed systems.

Data descriptions made in ASN.1 can be used for both SDL and TTCN specifications, thus making a tight integration between implementation and test, and promoting re-use.

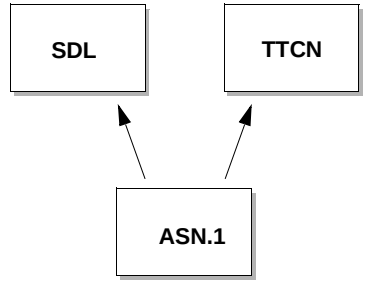

Figure 8: ASN.1 specifications can be shared between SDL and TTCN specifications

The Message Sequence Chart Language

History

During the last years, ITU has made a considerable effort in standardizing a formal language that defines *message sequence charts* (MSCs). A

first version of the MSC recommendation was published in the summer of 1992.

As defined in the recommendation Z.120, the MSC language offers a powerful complement to SDL in describing the dynamic behavior of an SDL system. Its graphical representation is well suited for presenting a complex dynamic behavior in a clear and unambiguous way that is easy to understand.

Theoretical Model

An MSC describes one or more traces from one node to another node of an abstract communication tree generated from an SDL specification. Basically, the information interchange is carried out by sending messages from one instance to another. In an SDL specification, those messages would coincide with the signals that are sent from one process and consumed in another process. The instances would correspond to any part of the specification (an SDL system, a block or a process).

Graphical and Textual Notations

The MSC language supports two notations that are equivalent. Beside the graphical notation (MSC/GR), a textual notation (MSC/PR) is standardized.

Application Areas

Among several application areas, we have selected the following:

- Producing documents with the purpose of defining the requirements on a system.
- Facilitating the design phase, by identifying and documenting a multitude of dynamic cases before starting designing with SDL.
- Presenting the execution of a simulation as a graphical output. This output is easy to follow and can later be verified against a reference. MSCs can be verified against an SDL system using the SDL suite.
- Presenting the execution trace of an SDL system during an interactive simulation and generation of reports.
- A convenient way to define test purposes, particularly in conjunction with the Autolink test generation features.

Chapter 1 Introduction to Languages and Notations